Mitigating traffic analysis attacks while maintaining on-path network observability

János Kövér[1,3][0009−0009−0088−6642], Roberto Guanciale[1,2][0000−0002−8069−6495], and György Dán[1,2][0000−0002−4876−0223]

[1] KTH Royal Institute of Technology, Dept. of Computer Science, Stockholm, Sweden
[2] Digital Futures
[3] Ericsson AB
{kover,robertog,gyuri}@kth.se

Abstract. Concealing distinguishing features in traffic patterns used in Traffic Analysis (TA) attacks also affects network observability and hence it is detrimental for legitimate traffic analysis (e.g., network monitoring, anomaly detection). The problem is particularly relevant in microservice-based cloud-native systems. In this paper we introduce a novel method that defends traffic flows against TA attacks and selectively exposes metadata to allow semi-trusted entities to recover certain traffic characteristics with low additional overhead by using a surplus area in the packets. In our architecture, proxies protect traffic between microservices using application-level logic and protocol features. We provide a PoC implementation and evaluation of the proposed method using the QUIC and HTTP/3 protocols for two network functions in the 5G Core Network and in a microservice benchmark application. We show that two events can be made indistinguishable for a storage channel attacker, while maintaining observability for a legitimate TA node. We also extend our defense to reduce the accuracy of a more powerful (timing channel) attacker by 20-30%.

Keywords: Traffic analysis · Network monitoring · Security and Privacy Protection · Network Protocols

1 Introduction

Despite the widespread use of encryption, networked systems remain vulnerable to information leakage through side channels exploited by traffic analysis (TA) attacks. Notable examples of TA attacks include website fingerprinting (WF) [17,45] and mobile application fingerprinting [29], which compromise confidentiality and privacy by applying statistical or Machine Learning (ML) techniques on traffic metadata. Cryptographic protocols, such as TLS 1.3 [40], IPsec ESP [28], QUIC [24] acknowledge TA threats, but lack built-in mitigation, leaving system designers responsible for mitigating side channels.

Due to the extensive adoption of the disaggregated architecture pattern, this work focuses on systems that rely on machine-to-machine interactions, such as

microservice-based applications, often using standardized deterministic protocols. For example, in the 5G Core Network (CN) the increasing disaggregation of Network Functions (NFs), together with their deployment in cloud infrastructures, introduces new attack surfaces for TA attacks. Deterministic protocols provide attackers with a predictable channel for information extraction, but they also allow for the design of provable and efficient defenses. Despite extensive research on mitigating TA attacks, existing proposals often lack formal guarantees (e.g., [43]), evaluation on real-world systems (e.g., [52]), or suffer from inefficiency (e.g., [18]).

Mitigating TA attacks is also at odds with network observability. Even when communications are encrypted, on-path measurement of coarse-grained traffic features, such as packet count and size distribution, remains valuable for statistical and ML-based anomaly detection [44]. On-path traffic analysis, via its simplicity and low impact on endpoints, contributes to network observability without requiring direct access to the underlying data. In this work, we introduce one of the first frameworks to counter TA attacks while preserving on-path network observability. We provide an implementation and evaluation of our defense (based on Supersequence [52] and Glove [36]), demonstrating its effectiveness against a wide range of attacks on real-world systems. We also address practical challenges of concealing side channels, specifically preventing storage channels arising from timing channels due to network stack behavior. Our work systematically identifies and diminishes exploitable patterns while maintaining usability and performance, making a significant step towards practical TA defenses.

Our mitigation framework performs traffic morphing through stateful proxies, ensuring that the defense remains fully transparent to the application. These proxies, placed within the trust boundary, have visibility of all packets and apply transformations to obscure traffic patterns. We utilize QUIC for efficient packet transport between proxies[4]. To preserve on-path network observability, we embed encrypted metadata after the UDP datagrams which allows trusted on-path monitors to recover crucial traffic information, such as packet sizes, without interfering with the end-to-end encryption of the communication. The proxies do not disrupt the application logic or require ad-hoc network protocols. We evaluate our defense against two types of attackers: those who can access only the storage channel and those with access to both the storage and timing channels. The evaluation demonstrates the effectiveness of our solution in reducing attacker accuracy (entirely preventing the attack in case of storage channels) while still allowing legitimate monitoring for operational purposes.

We demonstrate our framework on a real-world cloud-native 5G CN, where we defend communications between two key NFs. We also evaluate our solution on a microservice benchmark system that models a Media Service application, where we defend traffic between a microservice and a database.

The paper is organized as follows. In Section 2 we introduce the relevant concepts and notations. In Section 3 we describe the system model. Section 3.1 describes the attack model, while Section 4 describes the defense model. In Sec-

[4] QUIC has also been evaluated for use in 3GPP in [3].

tion 5 we describe the system design, and in Section 6 we evaluate the proposed defense. In Section 7 we discsus related work. Section 8 concludes the paper.

2 Background

Traffic Analysis techniques have been employed for both legitimate and malicious purposes. In this work, we implicitly assume that Traffic Analysis is performed on encrypted traffic, and makes use of packet lengths (and directions), packet ordering and inter-packet times as features.

To introduce the terminology, we borrow from the website fingerprinting domain [51]. Focusing on point-to-point communication between two parties, we use s to denote a sequence of packets $\langle p_1, p_2, ..., p_n \rangle \in S$, where $n = |s|$ is the sequence length and each packet $p \in \mathbb{R}_{\geq 0} \times \mathbb{Z}$ is a pair consisting of the interpacket time (compared to the previous packet in the sequence) and the packet's length. The direction of a packet is encoded as the sign of its length. We use s_t and s_l for sequences of the first and second component of packets, respectively.

We can formulate TA as a classification task. In a distributed system, a specific event in the finite set E triggers communication over the network. Depending on the granularity of the definition of events and due to variations, the same event can result in a set of different packet sequences distributed according to a certain probability distribution. We assume a finite and fixed precision for observing timing information. We define random variables X and Y on E and S respectively. Event X generates Y according to the conditional probability mass function $P_{Y|X}$. The goal of traffic analysis (TA) is to infer which event generated an observed packet sequence s.

When packets related to multiple events are sent over the same connection in parallel, individual sequences related to a single event are not easily identifiable in general. In this work, we assume events to be independent and that the attacker can isolate sequences per event [55]. We therefore focus on isolated sequences.

TA defense is generally implemented by transforming a packet sequence by a combination of the following operations: 1. Packet padding modifies packet sizes by adding removable bytes, ideally in an encrypted layer to hide the padding; 2. Sequence padding changes the number of packets, e.g., by sending dummies; 3. Packet scheduling controls the transmission times of packets. Further transformations include fragmentation and aggregation of individual packets.

3 System model

We consider a system illustrated in Fig. 1, consisting of a distributed application running in an environment where at least part of the network infrastructure is operated by a third party. The network traffic between the application instances is observed by a legitimate on-path traffic analysis node and a TA adversary.

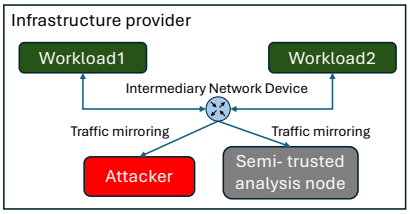

Fig. 1: Distributed application service in an infrastructure run by a third party

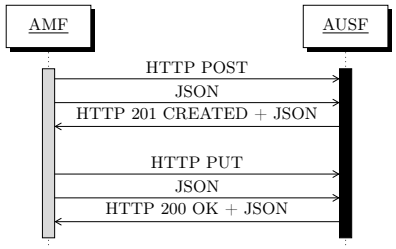

Fig. 2: Example message exchange during a successful UE registration

The owner of the distributed application (tenant) configures security controls (e.g., encryption of network traffic, etc.) to protect otherwise directly observable information. From a networking point of view, the tenant's trust boundary is at the (virtual) network interface of each application instance. However, communication metadata, such as size and timing of packets are visible outside the trust boundary too, e.g., to the infrastructure operator.

The distributed application consists of two separate workloads that communicate over a network. One of the applications receives requests from a source outside our model, which are handled by the two applications, involving encrypted network communication. The properties of a request, including its outcome, define events in the system, which are reflected as observable packet sequences on the network. The tenant may employ TA defense techniques to thwart unauthorized analysis of the traffic between applications.

The infrastructure provider also offers traffic mirroring for network observability, and the tenant uses this capability to perform traffic analysis to be able to detect changes in the behavior of the system (due to faults, etc.), in terms of communication patterns. The legitimate traffic analysis is carried out by a semi-trusted monitoring node that receives all traffic verbatim that is transmitted between the tenant's workloads but it does not have the cryptographic keys to access the tenant applications' payload. We assume the existence of a secure key provisioning scheme that can be used to install cryptographic keys (metadata keys) into the tenant workloads and the semi-trusted analysis node. These keys are used to enable metadata recovery for the semi-trusted analysis node even in the case of TA-defended traffic.

3.1 Threat model

We consider an adversary that can eavesdrop on the network communication between applications (e.g., the (network) infrastructure provider is honest-but-curious or breached). This model is equivalent to having access to mirrored traffic or to a packet capture file. The adversary cannot access the application payload data as cleartext. The goal of the adversary is to infer information about the tenant's operations by classifying observed packet sequences.

We consider two types of attackers: one that can only observe the storage channel (packet sequences without timing information, i.e., the order, size, and direction of packets) and one that has access to both storage and timing channels (i.e., packet sequences with timing information). The former attacker model is motivated by existing website fingerprinting attacks that only use the storage channel [31,32,45], and the possibility that timing information is imprecise or not available to the attacker (e.g., corrupted by noise, or data is only exposed through traffic summaries or device counters).

We assume that the attacker (1) is passive, i.e., does not change, delay, replay, drop or inject packets; (2) can associate IP addresses with particular endpoints (i.e, NFs or microservices); (3) knows the application layer protocol; (4) has access to labeled samples (packet sequences) from each event with and without TA defense applied; (5) knows how the TA defense is implemented; (6) has prior knowledge on the probability of events.

An attacker may use ML algorithms to perform the classification. If the TA-attacker can gather enough data, or can model the complete system, a classification strategy could be to classify each observation according to the event that more likely explains it (or randomly in case of equality).

3.2 Case studies

3GPP In the 5G CN, specialized functionalities are provided by Network Functions (NFs) that can be packaged to run as workloads in virtualized environments. As the first use case, we take NFs from a virtualized Core Network [6] that communicate directly[5] with each other. The communication between NFs is defined in the technical specification *5G System; Technical Realization of Service Based Architecture* [4]. Service Based Interfaces (SBI) use JSON in the application layer, transmitted over HTTP/2, (optionally [5]) TLS and TCP/IP.

In this work, we consider the Access and Mobility Management Function (AMF) and the Authentication Server Function (AUSF) NFs. The AMF handles NAS (Non-Access Stratum) messages sent by the user equipment (UE), and communicates with the AUSF to verify a UE's credentials. Fig. 2 shows an example message exchange. We consider two events, i.e., outcomes of user registrations: success and failure (indicated in the first response from AUSF), where the two outcomes are distinguishable by analysing the emanating traffic pattern between AMF and AUSF. Example sequences from one implementation (application data only, sizes only) for successful and failed registration: $s_{s_l} = \langle 61, 88, -339, 107, 46, -144 \rangle$ and $s_{f_l} = \langle 61, 88, -135 \rangle$, respectively. The vantage point of the attacker and the semi-trusted analysis node can be a cluster internal router, and we restrict the observable traffic to packets between AMF and AUSF. In our example, both the attacker and the semi-trusted analysis node want to identify different events, i.e., outcomes of UE registration procedures, by analysing the traffic between AMF and AUSF (i.e., maximizing the classification accuracy on the observed traffic).

[5] NFs can also communicate via a Service Communication Proxy (SCP), or in a roaming case, via a SEPP (Security Edge Protection Proxy).

Microservice benchmark As the second use case, we take an application from DeathStarBench [41,20], a collection of microservice-based applications modeling real-world systems. Most of the microservice interactions follow a simple request-response pattern through gRPC, Thrift, HTTP or a database-specific protocol. We use the Media Service application, where the user service interacts with the user database in the following way during a registration: it queries the database if there is an entry with the to-be-registered user name. If the response indicates that there is no such entry, the user service adds the new user to the database. In the end, this results in one or two request-response cycles (depending on the outcome of the registration attempt), with some of the packet sizes reflecting user-related metadata (e.g., length of user name). This is one of the more complex interactions in DeathStarBench. We consider the two outcomes of the registration attempts events. The attacker and semi-trusted analysis node have a similar purpose and vantage point as in the 3GPP case.

4 Defense model

The primary goal of the defense is to make events in the system (regardless of their probability of happening) indistinguishable based on the encrypted network communication between the workloads of interest. With E, S, X and Y defined as in Section 2, let $R_{Y|X}$ model the behavior of the TA-defended system. For any two events (e.g., x_1, x_2), let $P_Y(y) = R_{Y|X=x_1}$ and $Q_Y(y) = R_{Y|X=x_2}$ denote the two probability mass functions (PMFs) representing the distributions of the observable sequences. The two events can be considered indistinguishable if the Total Variation distance given by

$$\delta(P_Y, Q_Y) = \frac{1}{2} \sum_{y \in S} |P_Y(y) - Q_Y(y)|$$

is zero. If $\delta(P_Y, Q_Y) = 0$ for all pairs of events, we have $H(X|Y) = H(X)$.

We explore a particular confusion strategy, inspired by the bandwidth-efficient defenses described in [54,52,36], which transform sequences emanating from different events into the same sequence (or very similar ones).

In contrast, in our system, the defended traffic patterns are determined using knowledge of the well-defined application protocols, and TA defense is provided by proxies situated within the trust boundary, which are also responsible for cryptographic protection of the network traffic. Access to plaintext application messages enables the defense to apply fine-grained, application-specific transformation mechanisms. Importantly, during processing application messages, the defense can identify the corresponding event with certainty and also control packetization of application messages to make resulting patterns more deterministic.

The secondary goal of the defense is to enable authorized traffic analysis, so that a semi-trusted analysis node can still perform some form of analysis on the defended traffic after recovering certain original characteristics of the traffic. This may encompass recognizing patterns related to different events or anomalies.

5 Solution

Implementing TA defense requires that there is enough control to perform the necessary transformations. For the defense to be efficient, it needs to be able to track the application state to introduce only the necessary amount of overhead. Such efficient defense in a lower layer (e.g., the network layer) would need to re-implement some of the upper layer functionalities to parse and identify application messages. Moreover, a network layer defense would need to change the traffic flow while respecting the state of the transport protocol that runs on top of it. On the other hand, implementing the defense in an existing application protocol could require changes in the application and is therefore not desirable in general. Our solution[6] uses a combination of transport and application layer measures, implemented as proxies co-located with the NFs (microservices), similarly to the service-mesh pattern [30,48] as depicted in Fig. 3.

HTTP versions being semantically similar opens the way to use QUIC and HTTP/3 between the mitigation proxies. With a user-space QUIC implementation, the defense proxies have control over packetization and the use of UDP over IP enables selective metadata exposure.

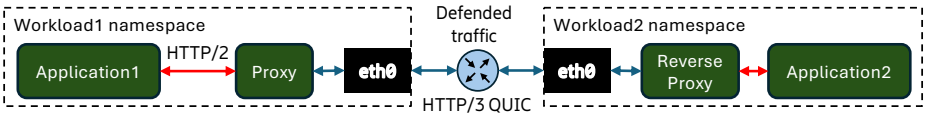

Fig. 3: TA mitigation proxies as side-cars.

The majority of message exchanges in 5G CN are in the form of request-response cycles, and the typical application-layer response sizes are 20-300 bytes [21]. Microservice communication in the Media Service system exhibits a similar behavior. Based on these observations, in each case, we can define a target (super)sequence that sequences generated by any event can be transformed into. In this work, we call sequence s a supersequence of sequences $s_1, ..., s_n$ iff any sequence s_i (where $1 \leq i \leq n$) can be transformed into s by some sequence of the following actions: packet padding, sequence padding and packet scheduling.

The defense intercepts application byte streams and constructs a single packet for each application message (request or response). Assuming that the application-layer protocol consists of a sequence of request-response cycles with dependencies between them, a supersequence for application-generated packets can be created during the defense design phase as follows. Based on the application protocol specification (and optionally logs), for each event (out of n events), and within each event, for every request and response we determine the worst-case execution (inter-message) time and resulting packet size. Encoding the direction of the message in the sign of the packet size (e.g., negative sign for re-

[6] PoC available at https://github.com/EricssonResearch/rev-pad

sponse), this results in sequences $s_1, ..., s_n$ (with $|s_m| \geq |s_j| \forall 1 \leq j \leq n$) of positive and negative sizes with their respective inter-message time. We then extend the shorter sequences with tuples $(0, 0)$ until the lengths are equal. A supersequence can then be obtained: $s = \langle ..., (max(\{s_{j_t}[i] : 1 \leq j \leq n\}), sign(s_{m_l}[i])max(\{|s_{j_l}[i]| : 1 \leq j \leq n\})), ... \rangle$. Ideally, transmitting the supersequence by applying the necessary transformations for different events would result in defended packet traces that are identical with respect to all observable features to an attacker as depicted on Fig. 4. (Note the added effects of the protocol (ACKs) in the resulting sequence on the wire.)

Fig. 4: Observable pattern during success and failure events in a defended system

5.1 Packet padding

We perform packet padding in the QUIC layer with PADDING frames that are transparent to the application. Each application request-response message exchange is transferred in a QUIC stream. To ensure we only add the necessary amount of padding, the upper layer informs the QUIC layer about the expected (worst-case) size for the packet carrying the particular stream with the message. Packets containing only ACK frames are padded to a fixed size.

5.2 Sequence padding

Since the two events we wish to make indistinguishable differ in the number of request-response cycles, sequence padding needs to be performed. As mimicking application exchanges would be a complicated task for the transport layer, this is done by performing dummy HTTP messages of very small size between the proxies, transparent to the real applications.

Minimizing impact on the storage channel A defense against attacks on the storage channel, which transforms only the number, order and size of application-level messages (i.e., implementing s_l only for a supersequence s), may allow leakage of timing information into the storage channel due to the interplay between the application (proxy) behavior and the protocol stack (QUIC). A manifestation of this can occur due to a QUIC implementation's acknowledgement strategy: For efficiency reasons, QUIC implementations should send ACK frames together with other frames if there are ack-eliciting packets that need to be acked [24]. If there are no other frames, e.g., carrying application data, to send, the stack will transmit a packet with a single ACK frame after some time

(ACK delay). Thus, depending on the sending time of the next packet (e.g., a response), there may be one or two packets visible on the network. In Appendix A we discuss this type of leakage further. As any packet that appears on the wire may contribute to information leakage, it may be worth to streamline the communication between the two applications. To make the packet sequences on the wire (emanating from application exchanges) short and deterministic, we configure the proxies and protocols in the following way: 1. A long-lived connection is used between the TA mitigation proxies 2. The dynamic table for QPACK is disabled 3. HTTP HEADER and DATA frames for the same message are sent in the same packet.

5.3 Packet scheduling

The TA mitigation proxies can perform scheduling of the HTTP request-response messages according to two strategies: app-scheduling-1 and app-scheduling-2.

App-scheduling-1 This strategy does not implement the prescribed worst-case timings in the supersequence, but only introduces enough delay before sending out a dummy request or response to make sure the acknowledgement is sent out for the previous packet, thereby emulating the NF (microservice) application behavior at least with respect to the ACK delay. The purpose here is to completely mitigate the storage channel leak caused by ACK multiplexing.

App-scheduling-2 This strategy schedules application-originated and sequence padding transmissions at the times prescribed in the supersequence. As an additional measure, during the processing of a sequence padding request, the reverse proxy performs a HTTP request to a local dummy server, to follow its behavior while handling a real application request more closely.

Hiding timing differences of packets with HTTP messages is not enough as the timing of stack-triggered packets (e.g., ones containing only an ACK frame) is not independent of the execution of application (proxy) code. As with some other, event-loop based implementations of QUIC (as documented in [35]), the execution of the protocol (e.g., for a particular connection) is blocked until the application yields control back to the event loop, which can serve as a source of timing information leak. In our case, different behaviors in the application layer (e.g., proxying, sleeping) were identifiable in the timing distribution of the acknowledgement (sent by the protocol stack) on a packet. As a simple solution, the (proxy) application layer delays processing of messages for a small duration so the sending time of an acknowledgement is not affected by the content of a message and ensuing computation. We call the combination of the above techniques "app-scheduling-2".

Protocol stack timers Timing of packets originated in the protocol stack are determined by the protocol implementation (without modification). However,

we implemented the more efficient ACK strategy, where ACK frames are sent together with other frames if there are ack-eliciting packets to acknowledge[7].

5.4 Selective exposure of metadata

To expose metadata selectively, we chose to carry the necessary data explicitly in each packet to help reconstruction of some aspects of what traffic patterns would look like without defense applied. We encode the extra information (to reconstruct packet sizes) after the UDP datagram, in a UDP Options [50] fashion. This information is encrypted using a key (metadata key) that is shared with the semi-trusted analysis node. The metadata key is independent from the TLS key used between the proxies, thus the semi-trusted analysis node does not breach the end-to-end confidentiality of the communication between the applications.

For our purposes, we want the metadata field to explicitly contain the amount of padding that was added to the packets. Since QUIC runs directly on top of UDP, adding the size of packet padding into the metadata field is straightforward. For simplicity, we assume that only one QUIC packet is sent in a UDP datagram.

6 Evaluation

To assess the effectiveness of our defense mechanism against TA attacks, we conducted a series of experiments using the applications described in Section 3.2. The evaluation primarily focuses on three aspects: defense against TA on the storage and timing channels and the feasibility of legitimate traffic analysis (network observability). Additionally, we evaluated the performance overhead introduced by our defense.

6.1 Experimental methodology

3GPP Our test environment consists of a 5G CN implemented with OPENAIR-CN-5G [37] and UERANSIM [8]. The TA mitigation proxy is based on aioquic [7] (with high limits on resources to avoid e.g., MAX_STREAMS frames) and h2 [15]. The 5G CN, RAN and UE simulators run in containers inside a single virtual machine where the AMF, AUSF and TA mitigation proxies do not share the CPU core with any other containerized workload, however, no other performance isolation technique is used.

User registrations are triggered sequentially (they do not overlap), and the outcomes (success, failure) are distributed randomly with an equal probability, while all packets are captured between AMF and AUSF. The true event is signaled out-of-band to help building the training and test data sets. After each 200th registration attempt, all the containers are re-started. Outlier sequences (that make up less than 0.5% of the dataset) where packet loss and retransmissions occurred are ignored in the evaluation.

[7] Sending ACK frames with other frames was observed in the vanilla implementation of the stack we use, but only with very specific application timing.

Microservice benchmark Similarly to the 3GPP case, the services in the DeathStarBench [41] Media Service system run in containers within a single virtual machine. To be able to use a generic HTTP API, we replaced the integrated database (MongoDB [2]) with CouchDB [1] and inserted our TA mitigation proxies (with slight adaptations) to hide the differences between the two events (register with existing and non-existing user name). Event generation and data collection are performed similarly to the 3GPP case. Note: To simplify the prototype implementation, we pad packet sizes (and timings) according to some assumed worst-case length of e.g. username. This means that our defense implicitly covers the case where different username lengths would be considered different events.

6.2 Defense against Traffic Analysis

Against the two types of adversaries presented in Section 3.1, we empirically evaluated several defense mechanisms: no defense, packet padding, sequence padding, app-scheduling-1 and app-scheduling-2, giving more details on the 3GPP system and only indicating the corresponding results for the microservice benchmark system.

In both experiments, we have $|E| = 2$ (success and failure, with equal probabilities), $P_Y(y) = R_{Y|X=success}$ and $Q_Y(y) = R_{Y|X=failure}$. Given enough data, or a suitable model, the TA attacker's best strategy would result in correct classification with probability $\frac{1}{2}(1 + \delta(P_Y, Q_Y))$.

Storage Channel Attacks We first evaluated our defenses against an attacker who can observe only packet lengths and their order in a sequence, i.e., s_l.

No defense The attacker can distinguish between the two events with certainty in both test systems, by e.g., relying on unique packet sizes in the first response ($s_{s_l}[3] < s_{f_l}[3]$ in the 3GPP case).

Packet padding. In both test systems, the attacker can distinguish between the two events with certainty, by looking at the sequence lengths ($|s_{s_l}| > |s_{f_l}|$).

Packet and sequence padding. Without any form of scheduling, the timing channel leaks into the storage channel (as described in section 5.2), causing the majority of the failed registrations to have a shorter sequence length, i.e., $|s_{s_l}| > |s_{f_l}|$. Based on empirical probability mass functions (PMFs) (on 600 samples), the accuracy for the best attacker is 0.94. The average accuracy of a decision tree classifier over 40 train-validate cycles with randomized sampling is 0.94. In the microservice benchmark system, the best attacker accuracy is 0.739, the accuracy of the decision tree classifier is 0.727 (based on 600 samples).

Packet and sequence padding with app-scheduling-1 or app-scheduling-2. After applying the defense, the events are indistinguishable on the storage channel.

Table 1: Estimation of best attacker accuracy per interpacket time at different resolutions

	app-scheduling-1				**app-scheduling-2**		
i	**10 µs**	**100 µs**	**1 ms**		**10 µs**	**100 µs**	**1 ms**
1	0.5	0.5	0.5		0.5	0.5	0.5
2	0.584	0.53	0.514		0.589	0.531	0.512
3	**0.999**	**0.994**	**0.99**		0.598	0.537	0.519
4	0.716	0.704	0.63		0.591	0.555	0.503
5	**1.0**	**1.0**	**0.999**		0.738	0.722	0.694
6	0.734	0.715	0.607		0.585	0.544	0.529
7	**1.0**	**1.0**	**1.0**		0.614	0.57	0.522
8	0.736	0.728	0.643		0.637	0.596	0.511

Note: (almost) perfect classification accuracies (caused by significant difference in application timing) are highlighted

Timing Channel Attacks Next, we evaluated our defenses against a more powerful attacker who can observe packet sizes, ordering and timing information. Combined with packet padding and sequence padding, we evaluate app-scheduling-1 and app-scheduling-2 separately, focusing only on interpacket time sequences s_t, since the events look identical on the storage channel.

The timestamps of the recorded individual packets have microsecond resolution, and the interpacket times take many different values. In our setup, it is infeasible to generate enough data to build an empirical PMF of interpacket time sequences (consisting of 8 values). Therefore, we estimate the best attacker accuracy for classifying each interpacket time in a sequence $s_t[i]$, where $1 \leq i \leq |s_t|$, independently, based on the statistical distance between their empirical PMFs obtained from reduced resolution datasets, where the recorded timestamps were rounded to the closest integer multiple of 10 µs, 100 µs and 1 ms.

Analysing classification accuracy of interpacket times that should be indistinguishable provides an indication of the accuracy being over-estimated due to insufficient amount of data, e.g., significant attacker advantage in classifying $s_t[2]$ (the timing of ACK on the first request) at a specific resolution is likely due to the empirical PMFs being too sparse and their support becoming disjoint.

Based on full sequences with microsecond resolution, we also perform 40 rounds of randomized experiments with a decision tree classifier (as adversary) and report the average attacker accuracy.

Packet and sequence padding, app-scheduling-1. According to our results on 4000 samples, the attacker can distinguish between the events with certainty for all the resolutions, since the support of the distribution of certain interpacket times is disjoint, e.g. $s_t[7]$, see Table 1. The average accuracy of the decision tree classifier on complete sequences was accordingly 1.0. Experiments on the Media Service benchmark system (600 samples) yielded the same result.

Packet and sequence padding, app-scheduling-2. Modifying the sending time of application-layer data clearly improves the defense according to our measurements (on 4000 samples), see Table 1. The average accuracy of the decision tree classifier on full sequences with µs resolution was 0.749. In the Media Service benchmark case, the accuracy of the decision tree classifier was 0.698 based on 800 samples.

6.3 Legitimate Traffic Analysis

We evaluated the ability of a semi-trusted analysis node to classify events while the defense was active. The analysis node was provided with the key needed to decrypt packet padding information. By extracting and analyzing this metadata, the node could accurately distinguish between successful and failed user registration events with certainty in both systems. The classification was based on the size of the first response message in each event, which uniquely identifies the outcome. Despite the defenses in place, the legitimate node could recover sufficient traffic characteristics to maintain full network observability.

Perfectly negating the effects of the defense, however, may not possible with this solution in general, but filtering out the packets pertaining to sequence padding (including ACKs) could be carried out by dropping short packets (after subtracting the padding size).

With certain defense strategies (e.g. traffic morphing-like) it may be possible to detect certain anomalies by analysing defended traffic without recovering original characteristics, but with e.g. constant time and constant size flooding-type defense it would probably be more difficult.

In real networks, disturbances may make it more difficult to perform analysis on the encrypted traffic. However, in case of re-ordering, certain features (e.g., unique packet sizes) may help re-aligning the packets and in case of a coarse analysis (e.g., histogram), order may not be significant. Moreover, findings in [34] suggest that CNN-based classifiers are invariant to packet position, hence reordering. Retransmissions may be also detected using unique packet sizes. Note that if the applications themselves use QUIC as transport protocol, an analysis node, independently of our defense, would have to deal with retransmissions and re-ordering, which are not directly identifiable to an on-path observer due to encryption.

6.4 Performance impact

Unlike defenses treating application messages as opaque byte stream, our defense can streamline the communication by multiplexing related HTTP HEADER and DATA frames (see client messages on Fig. 2) in one packet, resulting in fewer packets on the network. Additionally, it's possible to add only the necessary amount of padding to each packet. These give a clear advantage compared to Supersequence-based defenses using fixed padding [52,53], and multiplexing alone gives an edge over even variable-padding defenses similar to Glove [36] in comparable implementations, as shown in Table 2 for the 3GPP system where

Table 2: Network data overhead for the 3GPP system

	Undefended (success, HTTP/2, short-lived)	Undefended (failure, HTTP/2, short-lived)	Defended (fixed padding: Supersequence, Walkie-Talkie)	Defended (Glove)	Defended (our defense)
Bytes transferred	3723	1753	3560	2202	2008
Bytes transferred (only packets with app data)	1253	518	2856	1498	1304

we observed the undefended application to send HTTP HEADER and DATA separately in the requests over short lived connections. In the undefended microservice benchmark system (using HTTP/1.1) we didn't observe segmentation of requests or responses, however it may happen in general.

In the 3GPP case, the average end-to-end latency of user registration is 54 ms without defense and 128 ms with our defense. In the MediaService system the latency is 42 ms without defense and 124 ms with the defense. Other defenses using worst-case timings would have the same time overhead. The added latencies of only proxying without any defense were 21 ms and 17 ms in the two systems. With our defense applied, we only send 8 packets between the proxies during the event, leaving the network for tens of milliseconds unused between packets. This shows that the particular strategy used is efficient at least in this aspect compared to a constant-rate flooding defense (e.g., BuFLO discussed in Section 7).

Encrypting 4 bytes of data (e.g., metadata) incurs approximately 32 microseconds CPU overhead (as measured on a 13th Gen Intel(R) Core(TM) i7-1365U CPU), which is orders of magnitude less than the latency introduced by the timing channel defense.

7 Related work

Traffic analysis and web fingerprinting. Traffic analysis attacks using storage channels have long been a significant threat in various domains as shown in [31,23,38,26,45]. To our knowledge, none of the previous works point out the sensitivity of the storage channel to application timing changes and interactions with the transport protocol. We provide an in-depth analysis on the causes of observed information leaks not only in the application, but also in the transport protocol and how the two interact, giving insights in Section 6.2 into how application timing differences can lead to leakage via QUIC ACK frames. The importance of timing side channels, has been recently demonstrated by e.g., [42,39]. An in-application method fingerprinting appears in [29], highlighting the refining granularity of traffic analysis. Side channels have also been used to cluster and

subdivide packets in multiplexed communication into request-response pairs [55] or different flows [14].

Defenses. Regulation-based defenses aim to output highly similar patterns. Related works include constant-rate flooding-type defenses such as BuFLO [18] and its optimizations [12,13]. These approaches don't have full information about the sequences they treat and are therefore sub-optimal in resource usage. Traffic Morphing [54] defends the storage channel by outputting similar packet length distributions. Glove [36] proposes calculating and transmitting the same trace that can cover different website loads. The paper only contains simulation results. A similar approach in the Tor context (using direction information only) is discussed in [52] where covering sequences (Supersequences) are calculated for anonimity sets that are played when a corresponding site is loaded. Walkie-Talkie [53] enforces half-duplex communication to create bursts that are easier to merge into a supersequence. The design doesn't mitigate the channel related to the timing of requests. Defenses that calculate covering patterns and defend traffic without application-insight can't streamline the traffic (e.g., send request headers and body together) that would lower the number of timing features (interpacket times) available to the attacker. HTTPOS [33] is a browser-side WF attack mitigation. Without directly controlling the server's behavior, it's unclear how encrypted metadata could be included from the server side to maintain authorized observability. Furthermore, the proposed HTTP techniques are tailored for web browsing, and may not translate well to cases when HTTP is used to provide an API. A QUIC-oriented client-side defense framework is described in [46]. Palette [43] introduces a WF defense based on observed worst-case transmissions per timeslot. This is however sub-optimal (in the number of transmitted packets), and the following optimization step does not guarantee that all events will fit in the optimized output pattern, risking information leakage. Obfuscation defense approaches (e.g., [27,22]) inject dummy packets in a randomized manner to mask patterns.

None of these defenses provide an analysis on protocol-related effects of timing when applying the defense in a real implementation. Our approach applies a TA mitigation similar to [36,52] on top of QUIC with a pair of trusted, application-aware proxies, transforming interpacket times, as well as packet and sequence sizes. The main reason behind this choice over a constant-rate flooding or obfuscation-based defense is the lower network resource usage. We implemented the defense and performed an in-depth analysis on side channels. Furthermore, we combine our defense with a technique that allows network observability for authorized entities.

A model of the dependency between user behavior and resulting observable traffic features (in web applications) as an information theoretic channel appears in [9].

Legitimate traffic analysis. The notion of network anomaly can cover unusual patterns related to faults or other issues. In [49], any deviation from normal network behavior is considered a network anomaly in general. An overview on network anomaly detection with network intrusion detection systems (NIDS) in

focus is provided in [10]. A survey on ML-based encrypted traffic analysis (also concerning legitimate uses of TA) appears in [44]. Concrete legitimate TA approaches include detection of abnormal connections based on number of bytes sent by the endpoints [11], behavior learning to predict performance anomalies in virtualized systems [16], anomaly detection in edge cloud systems [19] and an anomaly detection system based on network flows [47]. In [25] the authors propose detecting security attacks based on RPC traces in DeathStarBench [41].

Many public cloud providers offer infrastructure for legitimate TA. There is also a plethora of commercial network observability and anomaly detection tools. Any effective TA defense will likely render legitimate TA approaches useless. Our solution alleviates this problem, however its applicability depends on the flexibility of the data collection mechanism that must be able to understand the metadata field to filter out at least some of the effects of the TA defense.

8 Conclusion

We propose a defense against adversarial TA that preserves partial traffic pattern visibility for authorized entities. Building on concepts from prior work [52,36], we implement the defense on top of the QUIC protocol. Our evaluation identifies application and proxy behaviors that can leak information under naïve defense implementations. By addressing these vulnerabilities, our system achieves full protection against storage channel attackers and partial protection against those with timing access.

We show that network observability for authorized entities is not obstructed completely even with the defense applied, by demonstrating a classifier that detects application-level events with certainty.

Our future work focuses on the following directions. We aim to explore automating the defense to adapt seamlessly to new applications and multi-class events. This includes dynamically learning supersequences through experimentation and static analysis. We also target scaling up and evaluating the defense on a large number of parallel events, focusing also on behavior e.g., in case of congestion, server-side resource constraints, etc.

Acknowledgements This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

A Leakage on the storage channel

In case a server (reverse proxy) responds after the ACK delay elapses, there will be two packets from the server visible to an observer (one carrying an ACK frame (acking the request packet) and the other carrying application data). On the other hand, if the server responds before the ACK is sent out, there will only be one packet leaving the server. Fig. 5 (note: ACK to the response is

omitted) depicts how timing information could leak into the storage channel due to multiplexing of ACK and STREAM frames. One way to overcome this type of problem could be that implementations enforce a consistent timing behavior in the application layer with respect to the protocol, as described in Section 5.3.

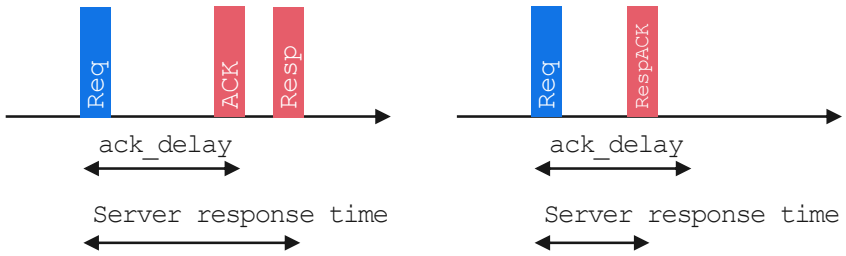

Fig. 5: Leakage from time to storage channel.

Another type of leakage was observed due to the sensitivity of packet ordering during the handshake in case short-lived connections are used for the request-response cycles. In particular, the HANDSHAKE_DONE message can be observed before or after the (client) application request packet, which observation may correlate with sensitive information (application behavior or system state).

References

1. Couchdb, https://couchdb.apache.org/
2. Mongodb, https://www.mongodb.com
3. 3GPP: Study on ietf quic transport for 5gc service based interfaces. Technical Report (TR) 29.893, 3rd Generation Partnership Project (3GPP) (9 2023), http://www.3gpp.org/DynaReport/29893.htm, version 18.0.0
4. 3GPP: 5G System; Technical Realization of Service Based Architecture; Stage 3. Technical Specification (TS) 29.500, 3rd Generation Partnership Project (3GPP) (2024), http://www.3gpp.org/DynaReport/29500.htm, version 19.0.0
5. 3GPP: Security architecture and procedures for 5G System. Technical Specification (TS) 33.501, 3rd Generation Partnership Project (3GPP) (2024), http://www.3gpp.org/DynaReport/33501.htm, version 19.0.0
6. 3GPP: System architecture for the 5G System (5GS). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP) (6 2024), http://www.3gpp.org/DynaReport/23501.htm, version 19.0.0
7. aioquic community: aioquic (2024), https://github.com/aiortc/aioquic
8. Ali Güngör: Ueransim (2024), https://github.com/aligungr/UERANSIM
9. Backes, M., Doychev, G., Köpf, B.: Preventing side-channel leaks in web traffic: A formal approach. In: NDSS (2013)

10. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: Methods, systems and tools. IEEE Communications Surveys & Tutorials **16**(1), 303–336 (2014)
11. Caberera, J., Ravichandran, B., Mehra, R.: Statistical traffic modeling for network intrusion detection. In: Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728). pp. 466–473 (2000)
12. Cai, X., Nithyanand, R., Johnson, R.: Cs-buflo: A congestion sensitive website fingerprinting defense. In: Proceedings of the 13th Workshop on Privacy in the Electronic Society. p. 121–130. WPES '14, Association for Computing Machinery, New York, NY, USA (2014)
13. Cai, X., Nithyanand, R., Wang, T., Johnson, R., Goldberg, I.: A systematic approach to developing and evaluating website fingerprinting defenses. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 227–238. CCS '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2660267.2660362, https://doi.org/10.1145/2660267.2660362
14. Chen, H.Y., Lin, T.N.: The challenge of only one flow problem for traffic classification in identity obfuscation environments. IEEE Access **9**, 84110–84121 (2021)
15. Cory Benfield and members of python-hyper: h2 (2024), https://pypi.org/project/h2/
16. Dean, D.J., Nguyen, H., Gu, X.: Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In: Proceedings of the 9th International Conference on Autonomic Computing. p. 191–200. ICAC '12, Association for Computing Machinery, New York, NY, USA (2012)
17. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In: 2012 IEEE Symposium on Security and Privacy. pp. 332–346 (2012). https://doi.org/10.1109/SP.2012.28
18. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In: 2012 IEEE Symposium on Security and Privacy. pp. 332–346 (2012)
19. Forough, J.: Machine learning for anomaly detection in edge clouds. Ph.D. thesis, Umeå University (2024)
20. Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al.: An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 3–18 (2019)
21. Gärdborn, P.: Is quic a better choice than tcp in the 5g core network service based architecture? (2020)
22. Gong, J., Wang, T.: Zero-delay lightweight defenses against website fingerprinting. In: 29th USENIX security symposium (USENIX security 20). pp. 717–734 (2020)
23. Herrmann, D., Wendolsky, R., Federrath, H.: Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security. p. 31–42. CCSW '09, Association for Computing Machinery, New York, NY, USA (2009)
24. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (May 2021). https://doi.org/10.17487/RFC9000, https://www.rfc-editor.org/info/rfc9000

25. Jacob, S., Qiao, Y., Ye, Y., Lee, B.: Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks. Computers & Security **118**, 102728 (2022). https://doi.org/https://doi.org/10.1016/j.cose.2022.102728
26. Juarez, M., Afroz, S., Acar, G., Diaz, C., Greenstadt, R.: A critical evaluation of website fingerprinting attacks. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 263–274. CCS '14, Association for Computing Machinery, New York, NY, USA (2014)
27. Juarez, M., Imani, M., Perry, M., Diaz, C., Wright, M.: Toward an efficient website fingerprinting defense. In: Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21. pp. 27–46. Springer (2016)
28. Kent, S.: IP Encapsulating Security Payload (ESP). RFC 4303 (Dec 2005). https://doi.org/10.17487/RFC4303, https://www.rfc-editor.org/info/rfc4303
29. Li, J., Zhou, H., Wu, S., Luo, X., Wang, T., Zhan, X., Ma, X.: FOAP: Fine-Grained Open-World android app fingerprinting. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1579–1596. USENIX Association, Boston, MA (Aug 2022)
30. Li, W., Lemieux, Y., Gao, J., Zhao, Z., Han, Y.: Service mesh: Challenges, state of the art, and future research opportunities. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). pp. 122–1225 (2019)
31. Liberatore, M., Levine, B.N.: Inferring the source of encrypted http connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. p. 255–263. CCS '06, Association for Computing Machinery, New York, NY, USA (2006)
32. Lu, L., Chang, E.C., Chan, M.C.: Website fingerprinting and identification using ordered feature sequences. In: Proceedings of the 15th European Conference on Research in Computer Security. p. 199–214. ESORICS'10, Springer-Verlag, Berlin, Heidelberg (2010)
33. Luo, X., Zhou, P., Chan, E.W., Lee, W., Chang, R.K., Perdisci, R., et al.: Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In: NDSS. vol. 11 (2011)
34. Luxemburk, J., Hynek, K., Čejka, T.: Encrypted traffic classification: the quic case. In: 2023 7th Network Traffic Measurement and Analysis Conference (TMA). pp. 1–10 (2023)
35. Microsoft: Msquic api (2024), https://github.com/microsoft/msquic/blob/main/docs/API.md
36. Nithyanand, R., Cai, X., Johnson, R.: Glove: A bespoke website fingerprinting defense. In: Proceedings of the 13th Workshop on Privacy in the Electronic Society. p. 131–134. WPES '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2665943.2665950
37. OpenAirInterface community: Openair-cn-5g (2024), https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed
38. Panchenko, A., Niessen, L., Zinnen, A., Engel, T.: Website fingerprinting in onion routing based anonymization networks. In: Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society. p. 103–114. WPES '11, Association for Computing Machinery, New York, NY, USA (2011)
39. Rahman, M.S., Sirinam, P., Mathews, N., Gangadhara, K.G., Wright, M.: Tik-tok: The utility of packet timing in website fingerprinting attacks. Proc. Priv. Enhancing Technol. **2020**(3), 5–24 (2020). https://doi.org/10.2478/POPETS-2020-0043, https://doi.org/10.2478/popets-2020-0043

40. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446, https://www.rfc-editor.org/info/rfc8446
41. SAIL Group at Cornell University: Deatstarbench (2019), https://github.com/delimitrou/DeathStarBench
42. Shapira, T., Shavitt, Y.: Flowpic: Encrypted internet traffic classification is as easy as image recognition. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 680–687 (2019)
43. Shen, M., Ji, K., Wu, J., Li, Q., Kong, X., Xu, K., Zhu, L.: Real-Time Website Fingerprinting Defense via Traffic Cluster Anonymization . In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3238–3256. IEEE Computer Society, Los Alamitos, CA, USA (May 2024)
44. Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q., Xu, K.: Machine learning-powered encrypted network traffic analysis: A comprehensive survey. IEEE Communications Surveys & Tutorials **25**(1), 791–824 (2023)
45. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 1928–1943. CCS '18, Association for Computing Machinery, New York, NY, USA (2018)
46. Smith, J.P., Dolfi, L., Mittal, P., Perrig, A.: QCSD: A QUIC Client-Side Website-Fingerprinting defence framework. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 771–789. USENIX Association, Boston, MA (Aug 2022)
47. Stoecklin, M.P., Le Boudec, J.Y., Kind, A.: A two-layered anomaly detection technique based on multi-modal flow behavior models. In: Claypool, M., Uhlig, S. (eds.) Passive and Active Network Measurement. pp. 212–221. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
48. the Istio Authors: Istio (2024), https://istio.io
49. Thottan, M., Ji, C.: Anomaly detection in ip networks. IEEE Transactions on Signal Processing **51**(8), 2191–2204 (2003)
50. Touch, D.J.D., Heard, C.M.: Transport Options for UDP. Internet-Draft draft-ietf-tsvwg-udp-options-33, Internet Engineering Task Force (Sep 2024), https://datatracker.ietf.org/doc/draft-ietf-tsvwg-udp-options/33/, work in Progress
51. Wang, T.: Website fingerprinting: Attacks and defenses. Ph.D. thesis, University of Waterloo (2016)
52. Wang, T., Cai, X., Nithyanand, R., Johnson, R., Goldberg, I.: Effective attacks and provable defenses for website fingerprinting. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 143–157. USENIX Association, San Diego, CA (Aug 2014)
53. Wang, T., Goldberg, I.: {Walkie-Talkie}: An efficient defense against passive website fingerprinting attacks. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1375–1390 (2017)
54. Wright, C.V., Coull, S.E., Monrose, F.: Traffic morphing: An efficient defense against statistical traffic analysis. In: NDSS. vol. 9 (2009)
55. Zhang, Q., Su, C.J.: Application-layer characterization and traffic analysis for encrypted quic transport protocol. In: 2023 IEEE Conference on Communications and Network Security (CNS). pp. 1–9 (2023)